Herramientas para Computación de Altas Prestaciones archivos MEX

archivos MEX: Introducción

Son programas en C, C++ o Fortran, compilados que se pueden ejecutar directamente desde Matlab

Se puede hacer para obtener mayor eficiencia o para reutilizar código ya existente

archivos MEX: Configuración

Para poder crear archivos mex necesitamos tener instalados compiladores compatibles con la versión de Matlab que estemos usando.

Compiladores soportados (Matlab R2014a)
http://www.mathworks.es/support/compilers/R2014a/index.html

Opciones:
Linux, al menos debería localizar los compiladores de GNU

Windows 32 bits, hay un compilador que viene con Matlab

archivos MEX: Configuración

Windows 64 bits: Si no hay otro compilador, se sugiere descargar Microsoft Windows SDK 7.1:

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=6b6c21d2-2006-4afa-9702-529fa782d63b&displaylang=en

que previamente requiere descargar e instalar el Framework .Net 4.0:
http://www.microsoft.com/download/en/details.aspx?id=17851

Averiguaremos si tenemos compiladores disponibles usando desde Matlab el comando

>>mex -setup

Archivos .mex en C:

Componentes de archivo . mex

-una o varias subrutinas "computacionales" en C, donde se hace los cálculos

-un "Gateway" o interfaz en C que se encarga de pasar de C a Matlab, fundamentalmente controlando el paso de parámetros. La idea es que podamos llamar a una función mex como si fuera una función Matlab normal.

-Una función Matlab puede tener varios argumentos de entrada y varios de salida:

Ejemplo: A la función **lu** se la puede llamar así:

>> [L,U,P] = lu(A,THRESH).

Archivos .mex en C:

Además, a una misma función Matlab se la puede llamar con un número variable de argumentos de entrada y/o de salida.

Ejemplo,a la función de Matlab **max** se la puede llamar de estas formas:

res=max(X) (Devuelve el maximo del vector X)

[res,ind]=max(X) (Devuelve el máximo del vector X y el índice en el que se encuentra)

res=max(X,Y) (Si se le pasan dos vectores del mismo tamaño X e Y, max devuelve un vector del mismo tamaño de X e Y, conteniendo los elementos más grandes cogidos de X o de Y).

Archivos .mex en C: Ejemplo “pordos”

Queremos convertir a mex la siguiente función (subrutina “computacional”)en C:

```
void pordos(double y[], double x[])
{
  y[0] = 2.0*x[0];
}
```

La llamada desde Matlab debería funcionar así:

```
>>pordos(4)

   Ans=8
```

Archivos .mex en C: Ejemplo “pordos”

Vamos a crear el archivo mex necesario.

Como es un ejemplo muy sencillo, vamos a meter la subrutina computacional y el gateway en el mismo archivo (pordos.c). En primer lugar, incluimos el archivo mex.h y ponemos la subrutina pordos

```
#include"mex.h"

void pordos(double y[], double x[])
{
  y[0] = 2.0*x[0];
}
...
```

Archivos .mex en C: Ejemplo “pordos”

A continuación empezamos la subrutina gateway: la cabecera de la función gateway es siempre igual:

```
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[] )
```

Argumentos del gateway:

-dos enteros nlhs y nrhs indicando con cuantos argumentos de entrada (nrhs) y de salida (nlhs) se ha llamado la función desde Matlab.

-arrays de punteros a los argumentos de salida (plhs) y a los argumentos de entrada (prhs).

El tipo mxArray, es un tipo de datos propio de Matlab que engloba a cualquier array de Matlab. plhs debe contener nlhs punteros, y prhs debe contener nrhs punteros.

Archivos .mex en C: Ejemplo “pordos”

Declaración de variables: En primer lugar, declaramos las variables que vamos a necesitar, dos punteros a reales de doble precisión x e y:

Además, para controlar errores en la llamada, necesitamos comprobar las dimensiones de las variables de entrada; lo hacemos con variables de tipo size_t (enteros) mrows y ncols.

La declaración de variables queda así:

```
....
double *x,*y;
size_t mrows,ncols;
...
```

Archivos .mex en C: Ejemplo “pordos”

En el caso de la rutina “pordos”, debemos comprobar que nrhs y nlhs deberían ser 1; en caso contrario, deberemos mostrar un mensaje de error con la función mexErrMsgIdAndTxt:

```
....
/* Check for proper number of arguments. */
if(nrhs!=1) {
   mexErrMsgIdAndTxt( "MATLAB:pordos:invalidNumInputs",
           "One input required.");
  } else if(nlhs>1) {
   mexErrMsgIdAndTxt( "MATLAB:pordos:maxlhs",
           "Too many output arguments.");
  }
...
```

Archivos .mex en C: Ejemplo “pordos”

Ahora comprobaremos las dimensiones del argumento de entrada, prhs[0]. Obtenemos las dimensiones con las funciones mxGetM y mxGetN. A continuación, comprobamos que prhs[0] es un número real de dimensiones 1 por 1, para ello comprobamos los valores de mrows y de ncols, así como el tipo de datos de prhs[0], con funciones mxIsDouble y mxIsComplex:

```
....
  /* The input must be a noncomplex scalar double.*/
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);

 if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
     !(mrows==1 && ncols==1) ) {
    mexErrMsgIdAndTxt(
"MATLAB:pordos:inputNotRealScalarDouble",
          "Input must be a noncomplex scalar double.");
  }
...
```

Archivos .mex en C: Ejemplo “pordos”

Para los argumentos de salida deberemos dimensionar las variables; habitualmente lo haremos con la función mxCreateDoubleMatrix, a la que le pasamos el número de filas y de columnas.

A continuación, creamos el “enlace”, mediante punteros, de los argumentos de entrada y salida de la rutina Gateway (prhs[0], plhs[0]) con los argumentos de la rutina computacional (x,y).

```
....

 /* Create matrix for the return argument. */
  plhs[0] = mxCreateDoubleMatrix((mwSize)mrows,
(mwSize)ncols, mxREAL);

/* Assign pointers to each input and output. */
  x = mxGetPr(prhs[0]);
  y = mxGetPr(plhs[0]);
...
```

Archivos .mex en C: Ejemplo “pordos”

Solo queda llamar a la subrutina computacional:

```
....

  /* Call the pordos subroutine. */

  pordos(y,x);
}
```

Una vez guardado el archivo pordos.c en el directorio de trabajo de Matlab, lo compilamos;

>> mex pordos.c

Al compilarlo se genera un archivo con el mismo nombre y con extensión mexw64.

Archivos .mex en C: Ejemplo “pordos”

Y lo ejecutamos:

```
>> pordos(6)
ans =

    12
```

También podemos comprobar lo que pasa si usamos un número equivocado de argumentos:

```
>> pordos(6,5)
Error using pordos
One input required.
```

Archivos .mex en C: Ejemplo "ArrayProduct"

El ejemplo ArrayProduct permite ver como trabajar con vectores y matrices en archivos. mex

```
void arrayProduct(double x, double *y, double *z, mwSize n)
{
    mwSize i;
    /* multiply each element y by x */
    for (i=0; i<n; i++) {
        z[i] = x * y[i];
    }
}
```

Archivos .mex en C: Ejemplo "ArrayProduct"

```
ncols = mxGetN(prhs[1]);

    /* create the output matrix */
    plhs[0] =
mxCreateDoubleMatrix(1,(mwSize)ncols,mxREAL);

    /* get a pointer to the real data in the output matrix */
    outMatrix = mxGetPr(plhs[0]);

    /* call the computational routine */

arrayProduct(multiplier,inMatrix,outMatrix,(mwSize)ncols);
```

Ejercicio Producto matrices

Crea un gateway para la siguiente subrutina que calcula el producto de matrices cuadradas:

```
void matprod(double *A, double *B, double *C, mwSize n)
{
   mwSize i,j,k;
   /* multiply each element y by x */
   for (j=0; j<n; j++)
       for(k=0;k<n; k++)
           for(i=0;i<n; i++)
     {
      C[i+j*n]= C[i+j*n]+ A[i+k*n]* B[k+j*n];
   }
}
```

Mex y OpenMP

-Aunque no oficialmente soportado, OpenMP funciona bien con los archivos mex. Para poder compilar programas mex con OpenMP, (de forma permanente) hay que editar el archivo mexopts.bat, en

\Users\$USER\AppData\Roaming\MathWorks\MATLAB\R2013a

y añadir a OPTIMFLAGS el flag /openmp

(Ojo 1)Este procedimiento es para Windows, hay un procedimiento análogo para Linux.

(Ojo 2) Si lo hacemos en el EVIR del DSIC, hay que volver a hacerlo cada vez que entramos . Una alternativa es usar CFLAGS para compilar con /openmp:

>> mex CFLAGS =‘ /openmp’ miprograma.c

Depuración de archivos mex Windows

Para depurar archivos mex en Windows necesitamos Visual Studio 2010 o 2012;

Procedimiento:

-Abre Matlab, compila tu archivo mex con la opción –g

-Sin cerrar Matlab, abre Visual Studio.

-En el menú “Herramientas”, selecciona la opción “Asociar a Proceso”.

-Busca el proceso Matlab y selecciónalo.

-Abre el archivo .c que contiene el código, pon un breakpoint (selecciona la línea donde quieres ponerlo, click en el botón derecho)

-Ve a Matlab y ejecuta el programa mex. Debe pararse en Visual Studio y a partir de ahí puedes depurar como cualquier otro programa en C.